VR Shooting Game

Team Members:

Name: Upre Ashish Kumar

Roll No : 21CSB0B62

Roles: Designer, Programmer, 3D artist, Debugger.

Name: Abhinash Kumar Shah

Roll No:21CSB0F37

Roles: Programmer, Debugger, Designer, 3D artist.

Department of Computer Science and Engineering

National Institute of Technology Warangal

Telangana, India – 506004

March 2025

Abstract

VR technology has been very influential in gaming industry. This is much in part due to the immense immersivity it provides to the user. This level of immersivity when in a shooter game will be very enjoyable to anyone who plays shooting games as it provides an actual first person shooter experience. This project presents a first person VR shooter game that consists of a ray gun that the player carries, enemies that spawn and shooting enemies down. Made with Unity 6, it uses the XR Device Simulator of XR Interaction toolkit to simulate an XR device so that we can develop the game without needing an actual XR device. The game consists of prefab based enemy generation and spawn at random locations, efficient collision handling and event driven scripting. The game delivers a compelling VR first person shooter experience, showcases the potential of Unity and serves as a foundation for developing more advanced VR games.

Table of Contents

6 Results & Discussion

6.1 Key findings and observations

6.2 Limitations of the project

6.3 Future enhancements

7 Conclusion

7.1 Summary of achievements

7.2 Future scope of the project

References

Appendices

1 Introduction

1.1 Background

The VR Shooting Game is a virtual reality experience developed using Unity Engine. It offers immersive gameplay, allowing players to use a ray gun to shoot enemies in a simulated VR environment. The game features realistic physics, dynamic enemy spawning, and interactive controls.

1.2 Problem statement

Traditional shooting games lack the immersive experience that VR technology offers. This project aims to create a realistic and interactive VR shooting environment, enhancing the gaming experience through real-time movement and spatial interaction.

1.3 Objectives and scope

- Develop a compelling VR-based shooting game.
- Implement ray gun mechanics for shooting bullets.
- Enable random enemy spawning with collision-based behavior.

- Utilize Unity XR Interaction Toolkit for realistic controls.

1.4 Significance of the project

This project demonstrates the potential of VR in gaming, providing:

- A realistic and engaging shooting experience.
- Hands-on learning of VR development and physics-based interactions.
- Potential application in VR training simulations.

2 Literature Review

2.1 Overview of VR/AR technology

VR technology creates a simulated environment using specialized hardware like VR headsets and motion controllers. AR overlays virtual objects on the real world, while VR offers fully immersive experiences.

2.2 Existing VR/AR applications related to the project

- **Beat Saber:** A VR rhythm game with dynamic sword-slashing mechanics.
- **Arizona Sunshine:** A VR zombie shooting game with realistic enemy AI.
- **Superhot VR:** A first-person shooter with time-controlled movement.

2.3 Comparative analysis with similar projects

Feature	Ours	Superhot VR	Arizona Sunshine
Graphics	Simple models and textures	Low-poly, minimalist style	Realistic graphics
Enemy AI	Basic collision-based logic	Time-controlled AI	Advanced zombie behavior
Gameplay	Point-and-shoot interaction	Bullet-dodging mechanics	Survival shooting
Technology	Unity XR Interaction Toolkit	Custom VR engine	Unity Engine
Spawning	Randomized enemy spawning	No spawning (static waves)	Scripted enemy waves

3 Methodology

3.1 Tools and technologies used

This project made extensive use of Unity 6 for its implementation. As we did not have a VR headset, we made use of XR Interaction toolkit package in Unity. XR Interaction Toolkit provides an XR Device Simulator that has a prefab of the same name that we have used in our project to simulate the game. XR Device Simulator is to be attached in the scene and it gives us the ability to control the player using controls given there itself on the screen.

Figure: *Interface provided by XR Device Simulaator of XR Interaction Toolkit*

The scripts present in Unity are written using C-Sharp (C#). So the logic present in the game is written in c#. The scripts are then attached to the Objects and they change the behaviour of that Object.

We made use of Visual Studio 2022 as our IDE of choice during the development due to its ease of use and seamless integration with Unity 6.

3.2 Development process

We have followed an Agile approach for the development of the project. We have brainstormed how the design is to be and started the development process. The process is done in iteration so that any changes that are to be made can be accommodated without freezing any design choices or requirements. We solved the problems we faced during development by debugging and using other approaches if the situation was un-debuggable. This also included a change in choice of technology we used well into the lifetime of our project.

3.3 System architecture & workflow

1. Hardware Components:

VR Headset & Controllers – Used for player interaction (shooting, aiming, movement).

PC (Development Machine) – Runs Unity, handles game logic, physics, and rendering.

2. Software Components:

Game Engine (Unity 6) – Manages rendering, physics, interactions, and animations.

XR Interaction Toolkit – Handles VR controller input and interactions.

C# Scripts – Implements game logic (shooting mechanics, enemy AI, movement, etc.).

3. Functional Modules:

A. Player Controller

RayGun.cs: Handles shooting, bullet instantiation, and input detection.

XR Rig (VR Player Setup): Controls player movement and camera view.

B. Enemy System

EnemySpawner.cs: Spawns enemies at random locations.

EnemyBehavior.cs: Controls enemy movement, reaction to bullets, and destruction.

C. Interaction System

VR Input Handling: Detects trigger presses for shooting.

Collision Detection: Registers when bullets hit enemies.

D. Game Environment

Ground Plane: Defines the playable area.

4 Implementation

4.1 Software and hardware setup:

Software Components:

- **Unity Engine:** Utilized as the primary development platform for its robust support in creating immersive VR experiences.

- **XR Interaction Toolkit:** Integrated to manage VR interactions, facilitating user input and object manipulation within the virtual environment.

- **Programming Languages:** Scripts were developed using C#, leveraging Unity's scripting API for game logic and interaction handling.

Hardware Components:

- **VR Headset**: The project was designed with VR headsets in mind providing users with an immersive experience.

- **Input Devices:** Motion controllers associated with the VR headset were employed to enable intuitive shooting mechanics and navigation within the game.

4.2 VR/AR environment and interaction design (UI/UX considerations)

- **Environment Design:**

The virtual environment was crafted to resemble a dynamic combat zone, incorporating elements such as obstacles and varied terrains to enhance realism. Attention was given to scale and spatial arrangement to ensure comfortable navigation and interaction.

- **User Interface (UI):**

The UI was designed to be minimalistic, displaying essential information like ammunition count and health status without obstructing the player's view. This approach aimed to maintain immersion while providing critical gameplay data.

- **User Experience (UX):**

To promote an intuitive user experience, the game employed natural interactions facilitated by the XR Interaction Toolkit. Players could aim and shoot using motion controllers, with haptic feedback providing tactile responses to actions, thereby enhancing engagement.

4.3 Features and functionalities

- **Realistic Shooting Mechanics:** Implemented a ray gun system that allows players to aim and fire projectiles accurately within the VR space.

- **Dynamic Enemy Behavior:** Developed enemy that reacts to player actions, including taking damage and being destroyed upon impact, adding depth to gameplay.

- **Randomized Enemy Spawning:** Designed a spawning system that introduces enemies at varied locations and intervals, ensuring unpredictable and challenging encounters

- **Interactive Environment:** Incorporated objects within the environment that players can interact with, enhancing the sense of presence and immersion.

4.4 Challenges faced and solutions

- We did not have a VR Headset which was challenging in the development process
- Initially we started using Meta XR all in one SDK for the development process
- It provides us a simulator to simulate quest headset
- But when we did some development and ran the game and checked, we observed that the simulator was not giving any output and simply showing a blank white screen
- This was un-debuggable as there wasn't any documentation for the version we were using
- We found out that there was a bug with unity 6 and hence changed unity version to 2022 & 2023 and checked, still the simulator wasn't working
- Hence we finally abandoned using meta all in one SDK and went forward with XR Interaction Toolkit that also provides a simulator called XR Device Simulator

5 Testing & Evaluation

5.1 Testing methods :

The VR Shooting Game underwent a comprehensive testing process to ensure its functionality, performance, and user experience. The testing methods employed included:

- **User Testing:** We both as users tested the project. The considerations were immersivity and responsivity of the project.

- **Performance Testing:** The game's performance was assessed by monitoring resource utilization during gameplay. This ensured that the game provied a smooth experience without lag or stuttering.

5.2 Performance analysis and User experience analysis:

The analysis focused on two primary aspects:

- **Performance:** The game consistently maintained a frame rate above 60 frames per second (FPS) on standard VR-capable hardware. Efficient memory management practices, such as destroying bullets and enemies after a set duration, contributed to this stability.

- **User Experience:** Feedback from user testing sessions highlighted the following:

→ **Immersion:** H level of immersion, attributing this to responsive controls and realistic enemy interactions.

→ **Controls:** With the XR Interaction Toolkit facilitating natural interactions within the VR environment, the controls were not that intuitive. This can just be considered another challenge due to lack of VR Headset.

→ **Motion Sickness:** Faced a little discomfort for long sessions.

5.3 Feedback and improvements

Based on the testing and evaluation phase, several areas for improvement were identified:

- **Enhanced Visuals:** Users suggested that more detailed models and textures would enhance the game's visual appeal and immersion.
- **Advanced Enemy AI:** Implementing more sophisticated enemy behaviors, such as strategic movement and varied attack patterns, could increase gameplay depth and challenge.
- **Expanded Content:** Introducing additional levels, enemy types, and weapons would provide greater variety and replayability.
- **Motion Sickness Mitigation:** To address discomfort, implementing features like adjustable movement speeds, teleportation options, and dynamic field-of-view adjustments during motion could help reduce motion sickness.

By integrating these improvements, the VR Shooting Game can offer a more engaging and comfortable experience for a broader range of players.

6 Results & Discussion

6.1 Key findings and observations

The VR Shooting Game successfully demonstrates an immersive and interactive shooting experience through the use of Unity Engine and the XR Interaction Toolkit. The project achieved the following key outcomes:

- **Realistic Shooting Mechanics:** The ray gun script allows smooth aiming and shooting interactions, providing a responsive and accurate firing experience.
- **Dynamic Enemy Behavior:** The implementation of collision-based AI ensures that enemies react to bullets, changing color upon impact and getting destroyed after a delay, adding a realistic touch.
- **Randomized Enemy Spawning:** The game uses a randomized enemy generation system, making each gameplay session unique and unpredictable.
- **Smooth VR Interaction:** The XR Interaction Toolkit enables fluid movement and interaction, enhancing the user's sense of presence in the virtual environment.
- **Optimized Performance:** The game maintains stable performance by destroying bullets and enemies after a delay, preventing memory overload and ensuring smooth rendering.

These observations highlight the project's effectiveness in creating an engaging VR shooting experience with consistent performance and interactive elements.

6.2 Limitations of the project

Despite its successful implementation, the VR Shooting Game has certain limitations:

- **Basic Graphics and Models:** The project uses simple models and textures, which reduces visual realism. More detailed assets could enhance the visual appeal.

- **Limited Enemy AI**: The collision-based AI is relatively basic, lacking advanced behaviors such as pathfinding or strategic movement.

- **Static Environment**: The game environment is static and lacks diversity, limiting the level of immersion. Adding more environmental elements or dynamic scenery could improve the overall experience.

- **No Multiplayer Mode:** The game is single-player only, reducing its replayability and engagement. Multiplayer functionality would offer a more competitive and interactive experience.

- **Limited Bullet Physics**: The bullet mechanics are simplified with basic velocity calculations. Implementing realistic bullet drop, ricochets, or physics-based interactions would make the shooting mechanics more realistic.

These limitations indicate areas for improvement and potential expansion in future versions of the game.

6.3 Future enhancements

To enhance the VR Shooting Game, several future improvements and features can be implemented:

- **Enhanced Graphics and Models:** Incorporating high-quality textures, realistic lighting, and detailed models will significantly improve the game's visual realism.

- **Advanced Enemy AI:** Implementing pathfinding algorithms (e.g., A* or NavMesh) will make enemy movements smarter and more dynamic, improving the challenge and engagement.

- **Expanded Gameplay Environment:** Adding multiple levels or dynamic environments will increase the game's replayability and provide a more diverse gaming experience.

- **Multiplayer Functionality:** Introducing a multiplayer mode will make the game more interactive and competitive, allowing players to engage in team-based or PvP shooting scenarios

- **Enhanced Bullet Physics:** Implementing realistic bullet trajectories, impact effects, and particle systems will create more engaging and believable shooting interactions.

- **User Interface (UI) Improvements:** Adding score counters, health bars, and game timers will enhance the player's experience by providing meaningful feedback during gameplay.
- **VR Haptics and Sound Effects:** Integrating haptic feedback and realistic sound effects will further immerse players in the VR environment.

These future enhancements will significantly improve the game's quality, realism, and replay value, making it more engaging and competitive.

7 Conclusion

7.1 Summary of achievements:

The project successfully implemented a functional VR shooting game using Unity and the XR Interaction Toolkit. It featured realistic ray gun mechanics, allowing players to shoot bullets with responsive interactions. The enemy AI was designed to respond to collisions by changing colour (red) and getting destroyed after a delay, creating a dynamic gameplay experience. The game incorporated random enemy spawning, ensuring varied enemy placement and replayability. The project showcased smooth and realistic interactions between the player and the game environment, providing an immersive experience.

7.2 Future scope of the project:

The project has potential for several enhancements:

i) Expanded Levels and Complex Terrains:

Adding new levels with diverse environments (e.g., forests, deserts) to make the game more engaging.

ii) Multiplayer Support:

Introducing multiplayer gameplay, allowing players to collaborate or compete in VR.

iii) Enhanced Enemy AI:

Implementing different attack patterns and behaviors for enemies, making them smarter and more challenging.

iv) Power-ups and Scoring System:

Adding collectible power-ups (e.g., increased bullet speed, health packs) and a scoring system to track player performance.

v) Graphics and Animations:

Improving the visuals with better textures, lighting effects, and animations for a more realistic experience.

References

- Unity - https://unity.com/
- XR Interaction Toolkit - https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html
- XR Device Simulator - https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/xr-device-simulator-overview.html
- Textures - https://assetstore.unity.com/publishers/4986
- Model - https://sketchfab.com/3d-models/50s-style-ray-gun-42b7288de197481292cc0f511f84a0dc
- Meta XR All In One SDK - https://developers.meta.com/horizon/downloads/package/meta-xr-sdk-all-in-one-upm/

Appendices

Code snippets:

- This is the script attached to the RayGun object.

```
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.XR.Interaction.Toolkit;
public class RayGun : MonoBehaviour
{
    public GameObject bulletPrefab;
    public Transform firePoint;
    public float bulletSpeed = 20f;
    public InputActionReference fireAction;

    void Update()
    {
        /**
         * shoot when trigger press is detected
         */
        if (fireAction.action.WasPressedThisFrame())
        {
            Shoot();
        }
    }

    void Shoot()
```

```
    {
        GameObject bullet = Instantiate(bulletPrefab, firePoint.position,
firePoint.rotation);
        Rigidbody rb = bullet.GetComponent<Rigidbody>();
        rb.linearVelocity = firePoint.forward * bulletSpeed;
        Destroy(bullet, 5f);
    }
}
```

- Below snippet is attached to Enemies, its for the behaviour of enemies
- It has the logic of what needs to happen when enemy is shot

```
using UnityEngine;

public class EnemyBehaviour : MonoBehaviour
{
    private Renderer enemyRenderer;

    void Start()
    {
        enemyRenderer = GetComponent<Renderer>();  // get renderer of enemy
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Bullet"))
        {
            enemyRenderer.material.color = Color.red;
            DestroyAfterDelay(2f);
        }
    }

    public void DestroyAfterDelay(float delay)
    {
        Invoke("DestroyEnemy", delay);
    }

    /**
     * turns red and gets destroyed
     */
    public void TakeDamage()
    {
        enemyRenderer.material.color = Color.red;
```

```
        DestroyAfterDelay(2f);
    }

    void DestroyEnemy()
    {
        Destroy(gameObject);
    }
}
```

- This script takes care of spawning enemies

```
using UnityEngine;

public class SpawnEnemies : MonoBehaviour
{
    public GameObject enemyPrefab;
    public Vector3 GroundCoordinates = new Vector3(0, (float)-0.2000003, 0);
    public float spawnInterval = 3f;
    public int maxEnemies = 5;
    private int enemyCount = 0;

    void Start()
    {
        InvokeRepeating(nameof(SpawnEnemy), 2f, spawnInterval);
    }

    void SpawnEnemy()
    {
        if (enemyCount >= maxEnemies)
            return; // Stop spawning if max reached

        Vector3 randomPosition = GetRandomSpawnPoint();
        Instantiate(enemyPrefab, randomPosition, Quaternion.identity);
        enemyCount++;
        spawnInterval += 1;
    }

    /**
     * gives a random point for enemy to spawn in the Ground
     */
    Vector3 GetRandomSpawnPoint()
    {
```

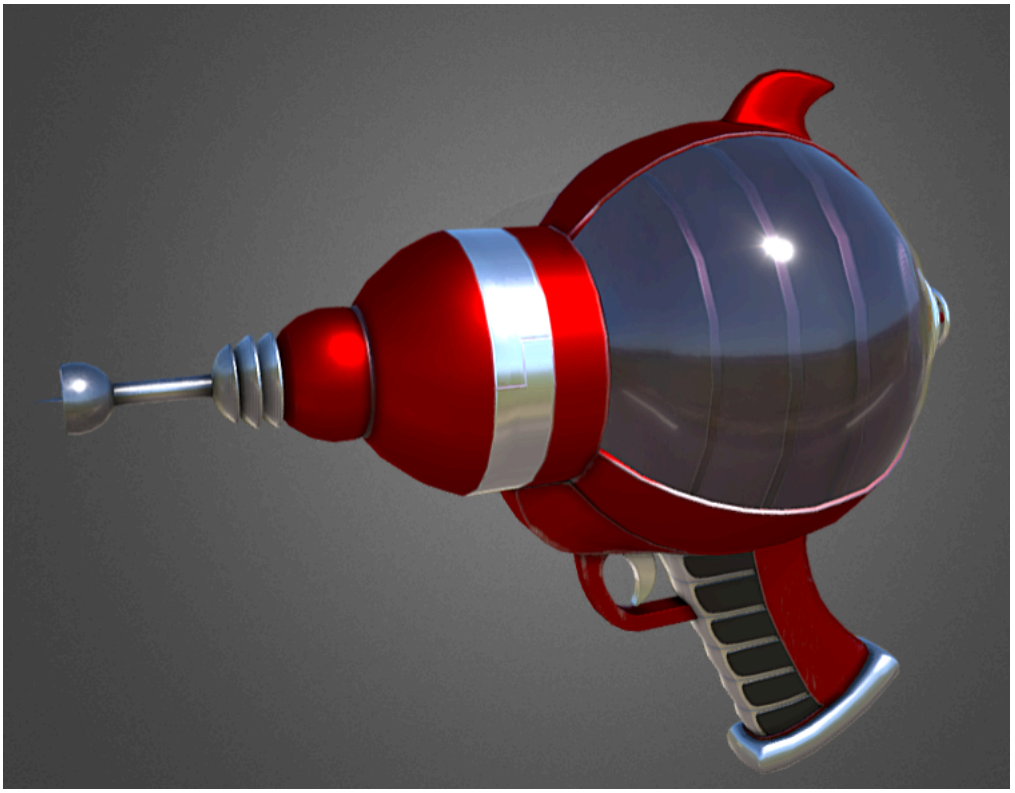
```
        float x = Random.Range(-5f, 5f);
        float z = Random.Range(-5f, 5f);
        float y = GroundCoordinates.y; // Keep enemies at ground level

        return new Vector3(transform.position.x + x, transform.position.y + y,
transform.position.z + z);
    }
}
```

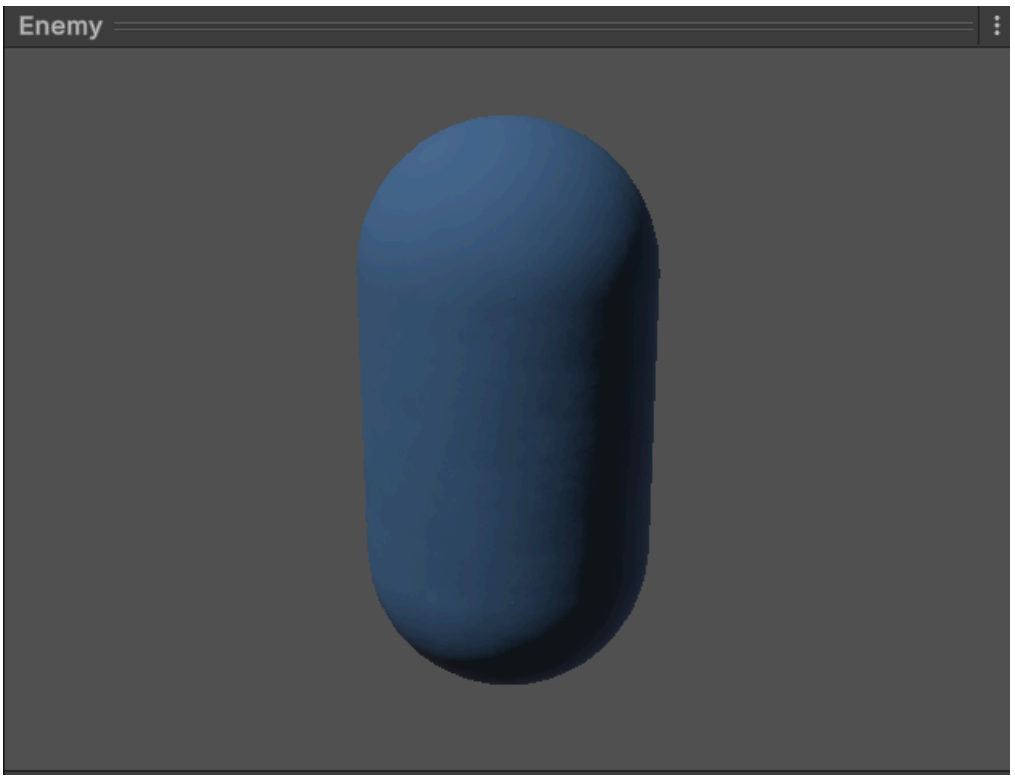
- UI of controls provided by XR interaction simulator

Models:

- Ray Gun:



- Enemy:



- Bullet